Student Name: **Student id:** **Sect#:** **Serial #:**

QUESTION #	1	2	3	4	TOTAL
MAX POINTS	16	12	18	14	60
POINTS EARNED					

University of Bahrain College of Information Technology
Department of Computer Science

ITCS332: Organization of Programming Languages **SECOND TEST** **Date: MAY 26, 2016**

****

QUESTION ONE: **[7+9 pts]**

a) What will be printed after executing the following C++ code?

```
20)     static int t=5;
21)     void funU()
22)     { static int x = 4;
23)       int f = x + t;
24)       int *p = new int(3);
25)       x = x + t - *p;
26)       t *= 2; *p = *p - 4;
27)       cout << x << '\t' << f << '\n';
28)       (*p)++;
29)       delete p;
30)     }
31)     int main()
32)     {
33)        t -= 2;
34)        funU();cout << t << '\t' ;
35)        funU();cout << t << '\t' ;
36)     }
```

```
4       7
6       7       10
12
```

b) Using the above given C++ code, answer the questions shown below.

1) The type of the variable ***x*** is ***static***
2) The type of the variable ***f*** is ***stack-dynamic***
3) The scope of the pointer variable **p** is ***from line #24 till line #30.***
4) The scope of the variable **t** is ***from line #20 till line #36.***
5) The variable pointed to by ***p*** is allocated memory on the ***heap*** and the variable ***f*** is allocated space on the ***stack*** .
6) The lifetime of a variable **x** begins when **the function *funU* is loaded first time** and ends when **the entire *program* terminates.**
7) The lifetime of a variable ***f*** begins whenever ***the function funU is called*** and ends whenever ***the function funU is terminated (return).***
8) The type of the variable (object) pointed to by **p** is ***explicit-heap dynamic***
9) The storage is bound to variable pointed to by ***p*** whenever ***the new operator in line #24 is executed*** and unbound whenever ***the delete statement in line#29 is executed.***

QUESTION TWO: **[8+4 pts]**

Part #1

1) The right side of a rule is called ***antecedent*** and the left side is called ***consequent***

2) In prolog, computations are performed by ***is*** operator and unification is done by **=** operator.

3) Prolog operates in ***Query*** or ***entry*** mode.

4) A clausal form of propositions contains ***conjunction*** operators in its right side and ***disjunction*** operators in its left side.

5) The prolog query `?- [2, f |U] = [2, f, j, 22, fi, tail].`
produces `U=[ j, 22, fi, tail ]`

6) The prolog query `?- [[red,Y]|X] = [[red, rat],[mouse,tom],cat].` produces
`Y = rat` and `X = [[mouse, tom], cat].`

```
myst9([],[]).
myst9([X],[Y]):- Y is 2*X.
myst9([X|Y], [X1|U]):- X1 is 2*X, myst9(Y,U).
```

7) The prolog query `?- myst9([4.5,3.75,7.5],M).` produces `M = [9,7.5, 15]`

Part #2

- Write a Prolog predicate(s) named **calc** that accepts a list of even number of any items and swaps the two adjacent elements in the list .

Sample queries:

```
?- calc([5,10,-9,13], S).
S = [10,5,13,-9]

?- calc([7,[10,5],20,25,[ali,isa],[1,2,3]], U).
U = [[10, 5], 7, 25, 20, [1, 2, 3], [ali, isa]].

?- calc([[a,b],[1,4],9,11,[x,y,f],[1,2,3],d,h], U).
U = [[1, 4], [a, b], 11, 9, [1, 2, 3], [x, y, f], h, d].
```

```
calc([],[]).
calc([X,Y|T],[Y,X|NT]) :- calc(T,NT).
```

QUESTION THREE: **[18 pts]**

a) **C++ code examples** **[1+2+2 pts]**

1) C++ provides different ways to create aliases. Give C++ code to create aliases.

```
int x = 99, *p1 = &x, *p2;
p2 = p1;
```

2) Give C++ code example to define a static length string.

```
char strU[20];
```

3) A variable may have multiple addresses at different places. Give C++ code that illustrates that.

```
void f1(...)
{   int x; ...      }

void main( )
{   int x; f1(…); ...     }
```

b) **Consider the following Ada-like code.** The values of n and m printed by **print(n,m):** **[6 pts]**

```
Procedure main is
  n,m: integer ;
  Procedure sub1 is
   begin
     n = n - 8; m *= 3;print(n,m);
   end ;
  Procedure sub2 is
    n : integer;
    begin
      n = 23 ; m = n - 7;sub1;
    end ;
begin
   n = 32 ; m = 11; sub2;
end ;
```

1) Under dynamic -scoped rules: ***n = 23-8=15***
m = 16*3=48

2) Under static -scoped rules: ***n = 32-8=24***
m = 11*3=33

c) **Fill in blanks** **[7 pts]**

1) Give one advantage and one disadvantage of using long names.
   a) The advantage is that they are **meaningful and improve the readability**.
   b) The disadvantage is that they are **wasting the memory space of the symbol table.**
2) The main disadvantage of stack-dynamic variables is large time overhead. Justify.
   Execution time is wasted for **allocating stack-dynamic variables** on every entry of the defining block and for **deallocating them on every exit** from the defining block.
3) The two main advantages of dynamic-length strings implemented as adjacent memory cells are:
   easy and fast string operations and **efficient use of memory space.**
4) The two major disadvantages of static variables are: **They do not support recursion**
   and **Low efficient use of memory space**.
5) A data type is a **collection of data values (objects)**
   and **a set of predefined operations that can be applied on those objects**
6) A decimal type is stored in memory in two forms: **packed BCD** and **unpacked BCD**.

QUESTION FOUR: Study the following C_like code and answer ALL questions below: **[14 pts]**

```
25) static int z[64];
26) void funA(double size)
27) {   int w[64];
28)     int f[size];
29)     int x[]={10,15,-24,-88,77};
30)     double *y = new double[24];
        ... ... ... ...
40)     z[5] = pow(z[10], z[6]);

        ... ... ... ...

60) }
61) void char funB()
62) {   static float d[10][20];
63)     char *uptr = new char[80];
64)     double sum; ... ... ... ...
99) }
```

1) The type of array ***x*** is ***fixed stack-dynamic.***
2) The type of array ***y*** is ***fixed-heap dynamic***
3) The type of array ***f*** is ***stack-dynamic***
4) The type of array ***z*** is ***static***
5) The array ***z*** is bound to storage during the ***loading*** time of the program. The call to the standard function ***pow*** in function ***funA*** is bound to the function's code at the ***linking*** time.
6) The lifetime of array **d** begins when ***the funB is called for the first time*** and ends when **the entire *program* terminates.**
7) The scope of a array ***x*** is ***from line #29 in function funA to line #60.***

8) The dynamic length strings are implemented using **Linked Lists** or **Adjacent memory Cells.**
9) The index type in array references is decided by **Language Designer.** The storage size allocated to each data type is decided by **Language Implementer.**
10) The code to access an array element is generated by the **Compiler.** Storing arrays using row- or column-major ordering of elements is decided by **Language Implementer.**
11) The two disadvantages of dynamic length string implemented as a linked list are: **complex and slow string operations** and **pointer space overhead.**
12) In C++, given the base address = `7500` and array `short U[100];`
    The address of the element `U[80]` = **7500 + (80-0)*2**
13) In C++, given the base address =`1240` and array `int F[20][100];`
    The address of element `F[11][66]` is **1240 + 4 *[(11-0)*100 + (66-0)]**